UCRL-WEB-201386

SLURM Reference Manual

Table of Contents

Preface

Scope: This manual explains the design goals and unique roles of LC's locally developed Simple Linux Utility for Resource Management (SLURM), intended as a customized replacement for RMS or NQS in allocating compute resources (mostly nodes) to queued jobs on machines running the CHAOS operating system. Sections describe the features of both control daemon SLURMCTLD and local daemon SLURMD, as well as SLURM's adaptability by means of plugin modules. The five SLURM user utilities for querying and controlling jobs managed by SLURM are also introduced. The features and options of SRUN, the tool used to launch both parallel interactive and batch jobs under SLURM management, receive especially detailed treatment.

Availability: SLURM is part of the CHAOS project, and is available on selected large LC clusters that run the CHAOS version of Linux.

Consultant: For help contact the LC customer service and support hotline at 925-422-4531 (open e-mail: lc-hotline@llnl.gov, SCF e-mail: lc-hotline@pop.llnl.gov).

Printing: The print file for this document can be found at:

```
on the OCF: http://www.llnl.gov/LCdocs/slurm/slurm.pdf
on the SCF: https://lc.llnl.gov/LCdocs/slurm/slurm_scf.pdf
```

Introduction

SLURM is LC's locally developed C-language Simple Linux Utility for Resource Management. SLURM is a job- and compute-resource manager that can run reliably and efficiently on Linux (CHAOS) clusters as large as several thousand nodes. Its features suit it to large-scale, high-performance computing environments, and its design avoids known weaknesses (such as inflexibility or fault intolerance) in available commercial resource management products for supercomputers.

This manual summarizes the specific service goals that SLURM was developed to meet, and explains the roles that it plays (relative to LC's Distributed Production Control System, for example) on LC production machines. Key to SLURM's operation are two software daemons: one (SLURMCTLD) controls the job queue and resource allocations, while the other (SLURMD) shepherds executing jobs on each compute node. Sections below explain the features and subsystems of each SLURM daemon. Additional sections tell how use of "plugin modules" make SLURM easily adaptable to many hardware situations, and introduce the five utility programs that give SLURM its direct user interface.

SRUN is the SLURM utility central to launching, assigning resources to, and guiding the execution of parallel jobs managed by SLURM, both interactively and through batch queues. Hence, the five ways to use SRUN (its "modes") and the often-elaborate interaction among the many SRUN options receive careful attention in several subsections devoted to that tool. SRUN also interacts with a set of special SLURM environment variables (like those used for job management by IBM's POE), explained in another subsection.

SLURM development is part of LC's larger CHAOS open-source operating system project, as explained in the separate CHAOS Reference Manual. (URL: http://www.llnl.gov/LCdocs/chaos) For a summary of known, significant differences between LC's Linux machines and those running AIX or Tru64 Unix, see the Linux Differences (URL: http://www.llnl.gov/LCdocs/linux) guide. And for general advice on managing (batch) jobs on LC production machines, consult the examples and comparisons in the basic EZJOBCONTROL (URL: http://www.llnl.gov/LCdocs/ezjobcontrol) guide.

SLURM Goals and Roles

SLURM Goals

SLURM was developed specifically to meet locally important criteria for a helpful, efficient way to manage compute resources on large (Linux/CHAOS) clusters. The *primary* threefold purpose of a cluster resource manager (such as LoadLeveler on LC's IBM ASCI machines or the Resource Management System (RMS) from Quadrics) is to:

- Allocate nodes--
  give users access (perhaps even exclusive access) to compute nodes for some specified time range so their job(s) can run.
- Control job execution--
  provide the underlying mechanisms to start, run, cancel, and monitor the state of parallel (or serial) jobs on the nodes allocated.
- Manage contention--
  reconcile competing requests for limited resources, usually by managing a queue of pending jobs.

At LC, an adequate cluster resource manager needs to meet two *general* requirements:

- Scalable--
  It must operate well on clusters with as many as several thousand nodes, including cases where the nodes are heterogeneous (with different hardware or configuration features).
- Portable--
  It must ultimately support jobs on clusters that have different operating systems or versions, different architectures, different vendors, and different interconnect networks. Linux/CHAOS is, of course, the intended first home for this software, however.

Any LC resource manager must also meet two *additional*, locally important, requirements:

- Compatible with DPCS--
  Since a resource manager is not a complex scheduler nor a complete batch system with across-cluster accounting and reporting features, it must support and work well within such a larger, more comprehensive job-control framework. At LC, the Distributed Production Control System (DPCS) (URL: http://www.llnl.gov/LCdocs/dpcs) provides that framework (see also the next section (page 7)).
- Compatible with QsNet--
  Since LC's Linux Project has already refined QsNet as its preferred high-speed interconnect for Linux/CHAOS clusters, an adequate resource manager must also allocate Quadrics QsNet resources along with compute nodes. But conversely, interconnect *independence* and the ability to easily support other brands of interconnect (such as Myrinet) is important too. Such independence allows great flexibility in pursuing new hardware configurations in future clusters.

Finally, to fit well into LC's emerging CHAOS environment, a resource manager should ideally have these three very beneficial *extra properties* as well:

- Fault Tolerant--
  Innovative scientific computing systems are often much less stable than routine business clusters, so a good local resource manager should recover well from many kinds of system failures (without terminating its workload), including failure of the node where its own control functions execute.
- Open Source--
  The software (source code) should be freely sharable under the GNU General Public License, as with other nonproprietary CHAOS components.
- Modular--
  An approach that clearly *separates* high-level job-scheduling functions from low-level cluster-administration functions allows for easier changes in scheduling policy without having to sacrifice working, familiar cluster-resource tools or features.

No commercial (or existing open source) resource manager meets all of these needs. So since 2001 Livermore Computing, in collaboration with Linux NetworX and Brigham Young University, has developed and refined the "Simple Linux Utility for Resource Management" (SLURM).

SLURM Roles

SLURM fills a crucial but mostly hidden role in running large parallel programs on large clusters.

Most users who run batch jobs at LC use job-control utilities (such as PSUB or PALTER) that talk to DPCS (the Distributed Production Control System), LC's locally designed *metabatch* system. DPCS:

- Provides a common user interface for batch-job submittal across all LC machines and clusters.
- Monitors resource use across machines and clusters.
- Implements bank-based fair-share scheduling policy, again, across all LC production machines.

To carry out its scheduling decisions, DPCS relies on the *native resource manager* on each machine or cluster where it assigns batch jobs to run. The basic duties of such a native resource manager are to:

- Get and share information on resource (chiefly node) availability.
- Allocate compute resources (chiefly, nodes or processors).
- Shepard jobs as their tasks execute.

On IBM machines, LoadLeveler serves as the native resource manager. On LC's nonIBM machines, DPCS relies on one of three other native resource managers to provide low-level job control:

- RMS (Resource Management System), used on "capability" clusters (devoted to one or two users at a time).
- NQS (Network Queueing System), used on "capacity" clusters (devoted to many simultaneous users).
- SLURM (newly introduced and still evolving to meet specific LC needs).

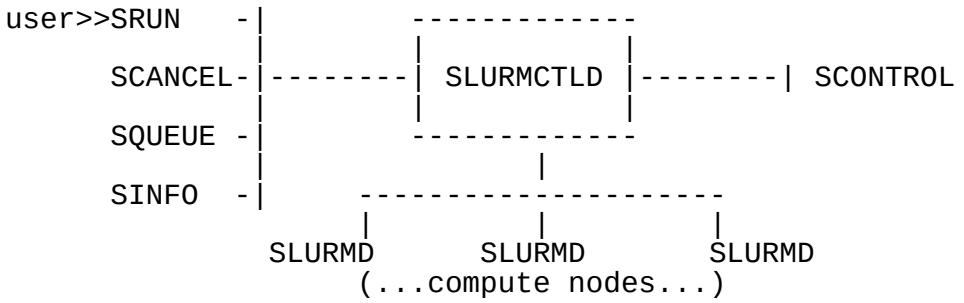
The key differences among these alternatives appear in this table:

	RMS	NQS	SLURM
Proprietary?	Yes	Yes	No, open source
Used on:	Machines with QsNet interconnect	Machines without QsNet	Either, interconnect independent
Suited for:	Capability clusters	Capacity clusters	Either with CHAOS
Node allocation:	Whole nodes allocated to jobs	Multiple jobs per node	Either possible

SLURM Features

SLURM Components

SLURM consists of two kinds of daemon (discussed here) and five command-line user utilities (next section (page 14)), whose relationships appear in this simplified architecture diagram:



SLURMCTLD

SLURM's central control daemon is called SLURMCTLD. Unlike the Portable Batch System daemon, SLURMCTLD is *multi*-threaded, so some threads can handle problems without delaying service to continuing normal jobs that also need attention. SLURMCTLD runs on a single management node (with a fail-over spare copy elsewhere for safety), reads the SLURM configuration file, and maintains state information on:

- nodes (the basic compute resource),
- partitions (logically disjoint sets of nodes),
- jobs (or resource allocations to run jobs for a time period), and
- job steps (parallel tasks within a job).

The SLURMCTLD daemon in turn consists of three software subsystems, each with a specific role:

Node Manager

> monitors the state and configuration of each node in the cluster. It receives state-change messages from each compute node's SLURMD daemon asynschonously, and it also actively polls those daemons periodically for status reports.

Partition Manager

> groups nodes into disjoint sets (partitions) and assigns job limits and access controls to each partition. The partition manager also allocates nodes to jobs (at the request of the Job Manager, below) based on job and partition properties. SCONTROL is the (privileged) user utility that can alter partition properties.

Job Manager — accepts job requests (from SRUN or a metabatch system like DPCS), places them in a priority-ordered queue, and reviews that queue periodically or when any state change might allow a new job to start. Qualifying jobs are allocated resources and that information transfers to (SLURMD on) the relevant nodes so the job can execute. When all nodes assigned to a job report that their work is done, the Job Manager revises its records and reviews the pending-job queue again.

SLURMD

The SLURMD daemon runs on every compute node of every cluster that SLURM manages and it performs the lowest level work of resource management. Like SLURMCTLD (above), SLURMD is multi-threaded for efficiency, but unlike SLURMCTLD it runs with root privilege (so it can initiate jobs on behalf of other users).

SLURMD carries out five key tasks and has five corresponding subsystems:

Machine Status

: responds to SLURMCTLD requests for machine state information and sends asynchronous reports of state changes to help with queue control.

Job Status
: responds to SLURMCTLD requests for job state information and sends asynchronous reports of state changes to help with queue control.

Remote Execution

: starts, monitors, and cleans up after a set of processes (usually shared by a parallel job), as decided by SLURMCTLD (or by direct user intervention). This often involves many process-limit, environment-variable, working-directory, and user-id changes.

Stream Copy Service

: handles all STDERR, STDIN, and STDOUT for remote tasks. This may involve redirection, and it always involves locally buffering job output to avoid blocking local tasks.

Job Control
: propagates signals and job-termination requests to any SLURM-managed processes (often interacting with the Remote Execution subsystem).

Portability (Plugins)

SLURM achieves portability (hardware independence) by using a general *plugin* mechanism. SLURM's configuration file tells it which plugin modules to accept.

A SLURM plugin is a dynamically linked code object that the SLURM libraries load explicitly at run time. Each plugin provides a customized implementation of a well-defined API connected to some specific tasks.

By means of this plugin approach, SLURM can easily change its:

- interconnect support (default is Quadrics QsNet).
- security techniques (default is to use crypto techniques to authenticate services to users and to each other).
- metabatch scheduler (default is LC's DPCS, with a "Grid" resource broker as an easy alternative).
- between-node communication "layers" (default is Berkeley sockets).

User Impact

The primary SLURM job-control tool is SRUN, (page 15) which fills the general role of PRUN (on former Compaq machines) or POE (on IBM computers). Your choice of run mode ("batch" or interactive) and your allocation of resources with SRUN strongly affect your job's behavior on machines where SLURM manages parallel jobs.

[Other specific user impacts coming soon here.]

SLURM Operation

SLURM Utilities

SLURM's five command-line utilities provide its direct interface for users (while DPCS utilities, as explained in EZJOBCONTROL (URL: http://www.llnl.gov/LCdocs/ezjobcontrol), provide an indirect interface). These utilities are:

SRUN	submits jobs to run under SLURM management. SRUN can (A) submit a batch job and then terminate, or (B) submit an interactive job and then persist to shepherd the job as it runs, or (C) allocate resources to a shell and then spawn that shell for use in running subordinate jobs. SLURM associates every set of parallel tasks ("job steps") with the SRUN instance that initiated that set.
SQUEUE	displays the queue of runing and waiting jobs (or "job steps"), including the JobId (used for SCANCEL), and the nodes assigned to each running job.
SINFO	displays a summary of available partition and node (*not* job) information (such as partition names, nodes/partition, and CPUs/node).
SCANCEL	cancels a running or waiting job, or sends a specified signal to all processes on all nodes associated with a job (only job owners or their administrators can cancel their jobs).
SCONTROL	(privileged users only) manages available nodes (for example, by "draining" jobs from a node or partition to prepare it for servicing).

SRUN (Submit Jobs)

SRUN Roles and Modes

SRUN executes tasks ("jobs") in parallel on multiple compute nodes at the same time (on machines where SLURM manages the resources). SRUN options let you both:

- Specify the parallel environment for your job(s), such as the number of nodes used, node partition, distribution of processes among nodes, and total time, and also
- Control the behavior of your parallel job as it runs, such as by redirecting or labeling its output, sending it signals, or specifying its reporting verbosity.

Because it performs several different roles, SRUN can be used in five distinct ways or "modes":

- SIMPLE.
  The simplest way to use SRUN is to distribute execution of a serial program (such as a UNIX utility) across a specified number or range of compute nodes. For example,

  ```
  srun -N 8 cp ~/data1 /var/tmp/data1
  ```

  copies (CP) file data1 from your common home directory into local disk space on each of eight compute nodes. This is very like running simple programs in parallel under AIX by using IBM's POE command (except that SRUN lets you set relevant environment variables on its own execute line, unlike POE). In simple mode, SRUN submits your job to the local SLURM job controller, initiates all processes on the specified nodes, and blocks until needed resources are free to run the job if necessary. Many control options can change the details of this general pattern.
- BATCH (WITHOUT DPCS/LCRM).
  SRUN can also directly submit complex scripts to the job queue(s) managed by SLURM for later execution when needed resources become available and when no higher priority jobs are pending. For example,

  ```
  srun -N 16 -b myscript.sh
  ```

  uses SRUN's -b option to place myscript.sh into the batch queue to later run on 16 nodes. Scripts in turn normally contain either MPI programs or other, *simple* invocations of SRUN itself (as shown above). SRUN's -b option thus supports basic, local batch service even on machines where LC's metabatch system DPCS/LCRM has not yet been installed (see below).
- ALLOCATE.
  To combine the job complexity of scripts with the immediacy of interactive execution, you can use SRUN's "allocate" mode. For example,

  ```
  srun -A -N 4 myscript.sh
  ```

  uses SRUN's (uppercase) -A option to allocate specified resources (here, four nodes), spawn a subshell with access to those resources, and then run multiple jobs using *simple* SRUN commands within the specified script (here, myscript.sh) that the subshell immediately starts to execute. This is very like allocating resources by setting AIX environment variables at the beginning of a script, and then using them for scripted tasks. No job queues are involved.

- ATTACH.
  You can monitor or intervene in an already running SRUN job, either batch (started with -b) or interactive ("allocated," started with -A), by executing SRUN again and "attaching" (-a, lowercase) to that job. For example,

  ```
  srun -a 6543 -j
  ```

  forwards the standard output and error messages from the running job with SLURM ID 6543 to the attaching SRUN to reveal the job's current status, and (with -j, lowercase) also "joins" the job so that you can send it signals as if this SRUN had initiated the job. Omit -j for read-only attachments. Because you are attaching to a running job whose resources have already been allocated, SRUN's resource-allocation options (such as -N) are incompatible with -a.

- BATCH (WITH DPCS/LCRM).
  On machines where LC's metabatch job-control and accounting system DPCS/LCRM is installed, you can submit (with the DPCS/LCRM utility PSUB) a script to DPCS that contains (simple) SRUN commands within it to execute parallel jobs later, after DPCS applies the usual fair-share scheduling process to your job and its competitors. Here DPCS/LCRM takes the place of SRUN's -b option for indirect, across-machine job-queue management.

SRUN SIGNAL HANDLING.

Signals sent to SRUN are automatically forwarded to the tasks that SRUN controls, with a few special cases. SRUN handles the sequence CTRL-C differently depending on how many it receives in one second:

```
CTRL-Cs within one second
-------------------------
First    reports the state of all tasks
         associated with SRUN.
Second   sends SIGINT signal to all
         associated SRUN tasks.
Third    terminates the job at once,
         without waiting for remote tasks
         to exit.
```

MPI SUPPORT.

On computer clusters with a Quadrics interconnect among the nodes (such as Adelie and Emperor on SCF, or MCR and ALC on OCF) SRUN directly supports the Quadrics version of MPI without modification. Applications built using the Quadrics MPI library communicate over the Quadrics interconnect without any special SRUN options.

You may also use MPICH on any computer where it is available. MPIRUN will, however, need information on its command line identifying the resources to use, namely

```
-np SLURM_NPROCS -machinefile filename
```

where SLURM_NPROCS is the environment variable that contains the (-n) number of processors to use and *filename* lists the names of the nodes on which to execute (the captured output from /bin/hostname run across those nodes with simple SRUN). Sometimes the MPICH vendor configures these options automatically.

Comparison with POE

SRUN and AIX's POE (Parallel Operating Environment) both use UNIX environment variables to manage the resources for each parallel job that they run. Of course, variables with comparable roles have different names under each system (and both systems have many other environment variables for other purposes too).

Two *differences* in detail between environment-variable use by SRUN and POE are noteworthy:

- SRUN assigns values to its resource-management variables by means of its own interactive options, one option for each environment variable (plus extra control options, such as -j). Instead, POE uses the usual SETENV or EXPORT utilities to assign values to its environment variables.
- POE's LoadLeveler ignores many environment variables when it run batch jobs under AIX on LC machines. SLURM does not ignore the corresponding environment variables when set by SRUN, even for batch runs.

This chart lists the SLURM (SRUN-set or inferred) resource-management environment variables for which direct POE counterparts exist. For an explanatory inventory of all SLURM environment variables, see the separate section (page 23) below.

Environment Variable Role	SRUN Option To Set	SLURM Variable Name	POE Variable Name
Total processes to run	-n	SLURM_NPROCS	MP_PROCS(*)
Total nodes allocated	-N	SLURM_NNODES	MP_NODES(*)
Node list for this job	(inferred)	SLURM_NODELIST	MP_SAVEHOSTFILE(*)
MP ID of current process	(inferred)	SLURM_PROCID	MP_CHILD
Output mode choice	-o (lc)	SLURM_STDOUTMODE	MP_STDOUTMODE
Partition for this job	-p	SLURM_PARTITION	MP_RMPOOL(*)
Debug message level	-d	SLURMD_DEBUG	MP_INFOLEVEL
Output message level	-v (lc)	SLURM_DEBUG	MP_INFOLEVEL
Output label choice	-l	SLURM_LABELIO	MP_LABELIO

(*)Ignored by LoadLeveler for batch jobs on AIX machines at LC.

SRUN Run-Mode Options

For a strategic comparison (with examples) of the five different ways to use SRUN, see "SRUN Roles and Modes," above. (page 15) This section explains the *mutually exclusive* SRUN options that enable its different run modes. Each option has a one-character (UNIX) and a longer (Linux) alternative syntax.

-b (--batch) runs a script (whose name appears at the end of the SRUN execute line, *not* as an argument to -b) in batch mode. You cannot use -b with -A or -a.

RESULT.
SRUN copies the script, submits the request to run (with your specified resource allocation) to the local SLURM-managed job queue, and ends. When resources become available and no higher priority job is pending, SLURM runs the script on the first node allocated to the job, with STDIN redirected from /dev/null and STDOUT and STDERR redirected to a file called *jobname*.out in the current working directory (unless you request a different name or a more elaborate set of output files by using -J or -o). In other words, -b executes the script through a queue, unhooked from terminal interaction, but not under DPCS/LCRM control.

SCRIPT REQUIREMENTS.
(1) You must use the script's absolute pathname or a pathname relative to the current working directory (SRUN ignores your search path).
(2) SRUN interprets the script using your default shell unless the file begins with the character pair #! followed by the absolute pathname of a valid shell.
(3) The script must contain MPI commands or other (simple) SRUN commands to initiate parallel tasks.

-A (uppercase, --allocate)

allocates compute resources (as specified by other SRUN options) and starts ("spawns") a subshell that has access to those allocated resources. No remote tasks are started. You cannot use -A with -b or -a.

SCRIPTED USE.
If you specify a script at the end of SRUN's execute line (*not* as an argument to -A), the spawned shell executes that script using the allocated resouces (interactively, without a queue). See the -b option for script requirements.

UNSCRIPTED USE.
If you specify no script, you can then execute other instances of SRUN interactively, within the spawned subshell, to run multiple parallel jobs on the resources that you allocated to the subshell. The resources (nodes, etc.) will only be freed for other jobs when you terminate the subshell.

-a *jobid* (lowercase, --attach=*jobid*)

attaches (or reattaches) your current SRUN session to the already running job whose SLURM ID is *jobid*. The job to which you attach must have its resources managed

by SLURM, but it can be either interactive ("allocated," started with -A) or batch (started with -b). This option allows you to monitor or intervene in previously started SRUN jobs. You cannot use -a with -b or -A. Because the running job to which you attach already has its resources specified, you cannot use -a with -n, -N, or -c. You can only attach to jobs for which you are the authorized owner.

READ-ONLY.
By default, -a attaches to the designated job read-only. STDOUT and STDERR are copied to the attaching SRUN, just as if the current SRUN session had started the job. However, signals are *not* forwarded to the remote processes (and a single CTRL-C will detach the read-only SRUN from the job).

READ-WRITE.
If you use -j (--join) or -s (--steal) along with -a, your SRUN session "joins" the running job and can also forward signals to it as well as receive STDOUT and STDERR from it. If you join a SLURM batch (-b) job, you can send signals to its batch script. Join (-j) does not forward STDIN, but steal (-s, which closes other open sessions with the job) does forward STDIN as well as signals.

-j (--join) joins a runing SLURM job (always used only with -a, --attach, to specify the *jobid*). This not only duplicates STDOUT and STDERR to the attaching SRUN session, but it also forwards signals to the job's script or processes as well.

-s (--steal) steals all connections to a running SLURM job (always used only with -a, --attach, to specify the *jobid*). STEAL closes any open sessions with the specified job, then copies STDOUT and STDERR to the attaching SRUN session, and it also forwards *both* signals and STDIN to the job's script or processes.

SRUN Resource-Allocation Options

These SRUN options (used alone or in combination) assign compute resources to your parallel SLURM-managed job. Each option has a one-character (UNIX) and a longer (Linux) alternative syntax.

-n *procs* (lowercase, --nprocs=*procs*)

requests that SRUN execute *procs* processes. To control how these processes are distributed among nodes and CPUs, combine -n with -c or -N as explained below (default is one process per node).

-N *n* (uppercase, --nodes=*n*)

allocates at least *n* nodes to this job, where *n* may be either
(1) a specific node count (such as -N 16), or
(2) a hyphen-separated range from a minimum to a maximum node count (such as -N 2-4).
If the nodes are partitioned, each partition's node limits supersede those specified by -N (jobs that request more nodes than the partition allows never leave the PENDING state). To change partitions, use SRUN's -p option (below). Combinations of -n and -N control how job processes are distributed among nodes according to the SRUN policies listed here:

-n/-N COMBINATIONS.
SRUN infers your intended number of processes per node if you specify *both* the number of processes and the number of nodes for your job. Thus -n 16 -N 8 normally results in running 2 processes/node (but see the next policy for exceptions).

MINIMUM INTERPRETATION.
SRUN interprets all node requests as minimum node requests (so -N 16 means "at least 16 nodes"). If some nodes lack enough CPUs to cover the process count specified by -n, SRUN will automatically allocate more nodes (than mentioned with -N) to meet the need (for example, if not all nodes have 2 working CPUs, then -n 32 -N 16 together will allocate *more than* 16 nodes so that all processes are supported). The actual number of nodes assigned (not the number reqested) is stored in environment variable SLURM_NNODES.

CPU OVERCOMMITMENT.
By default, SRUN never allocates more than one process per CPU. If you intend to assign multiple processes per CPU, you must invoke SRUN's -O (uppercase oh) option along with -n and -N (thus -n 16 -N 4 -O together allow 2 processes/CPU on the 4 allocated 2-CPU nodes).

INCONSISTENT ALLOCATION.
SRUN rejects as errors inconsistent -n/-N combinations. For example, -n 15 -N 16 requests the impossible assignment of 15 processes to 16 nodes.

-c *cpt* (lowercase, --cpus-per-task=*cpt*)

assigns *cpt* CPUs per process for this job (default is one CPU/process). This option supports multithreaded programs that require more than a single CPU/process for best performance.

-n/-c COMBINATIONS.
For multithreaded programs where the density of CPUs is more important than a specific node count, use both -n and -c on the same SRUN execute line (rather than -N). Thus -n 16 -c 2 results in whatever node allocation is needed to yield the requested 2 CPUs/process. This is the reverse of CPU overcommitment (see -N and -O).

-p *part* (lowercase, --partition=*part*)

requests nodes only from the *part* partition (the default partition is assigned by the system administrator on each separate LC machine).

-t *min* (lowercase, --time=*min*)

allocates a total of *min* minutes for this job to run (default is the current partition's time limit). If *min* exceeds the partition's time limit, then the job never leaves the PENDING state. When the time limit has been reached, SLURM sends each job process SIGTERM followed (after a pause specified by SLURM's KillWait configuration parameter) by SIGKILL.

-T *nthreads* (uppercase, --threads=*nthreads*)

requests that SRUN allocate *nthreads* threads to initiate and control the parallel tasks in this job (default is the smaller of 10 or the number of nodes actually allocated, SLURM_NNODES).

SRUN Control Options

[Details coming soon.]

SRUN I/O Options

[Details coming soon.]

SRUN Constraint Options

[Details coming soon.]

Environment Variables

Many SRUN options have corresponding environment variables (analogous to the approach used with POE). The SRUN option, if invoked, always overrides (resets) the corresponding environment variable (which contains each job feature's default value, if there is a default).

In addition, SRUN sets these environment variables for each executing task on the remote compute nodes:

SLURM_JOBID

specifies the job ID of the executing job.

SLURM_NODEID

specifies the relative node ID of the current node.

SLURM_NODELIST

specifies the list of nodes on which the job is actually running.

SLURM_NPROCS

specifies the total number of processes in the job.

SLURM_PROCID

specifies the MPI rank (or relative process ID) for the current process.

Disclaimer

Keyword Index

To see an alphabetical list of keywords for this document, consult the next section (page 26).

Alphabetical List of Keywords

Keyword	Description
a	The alphabetical index of keywords.
availability	Where SLURM runs.
constraint-options	Specifying job constraints.
control-options	Managing general job features.
date	The latest changes to this document.
entire	This entire document.
environment-variables	SRUN environment variables defined.
i-o-options	Handling job input, output, errors.
index	The structural index of keywords.
introduction	Overview of SLURM features, comparisons.
poe-comparison	SRUN and (IBM) POE differences.
portability	How plugin modules make SLURM adaptable.
resource-allocation	Assigning compute resources to jobs.
revisions	The complete revision history.
run-mode-options	Enabling different job-run alternatives.
scope	Topics covered in this document.
slurm-components	Software units comprising SLURM.
slurm-features	How SLURM works.
slurm-goals	SLURM design goals as resource manager.
slurm-operation	User interaction with SLURM.
slurm-roles	RMS, NQS, DPCS, SLURM compared.
slurm-strategy	Special benefits built into SLURM.
slurm-utilities	SLURM's direct user utility programs.
slurmctld	SLURMCTLD control daemon explained.
slurmd	SLURMD local daemon explained.
srun	Job-submittal and resource utility.
srun-roles	SRUN roles and modes compared.
title	The name of this document.
user-impact	SLURM's effect on typical jobs.
who	Who to contact for assistance.

Date and Revisions

Revision Date	Keyword Affected	Description of Change
21Oct03	srun	Major SRUN features, options explained.
	introduction	SRUN's central role introduced.
	index	Nine new keywords for new sections.
20Aug03	entire	Draft edition of SLURM manual.

TRG (21Oct03)

UCRL-WEB-201386

TRG (21Oct03) Contact on the OCF: lc-hotline@llnl.gov, on the SCF: lc-hotline@pop.llnl.gov